The Journal of Systems and Software 126 (2017) 45–56

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Development and use of a new task model for cyber-physical systems: A real-time scheduling perspective

Jinkyu Lee[a,*], Kang G. Shin[b]

[a] Sungkyunkwan University (SKKU), Suwon, South Korea
[b] Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, U.S.A.

ARTICLE INFO

Article history:
Received 19 June 2016
Revised 21 December 2016
Accepted 12 January 2017
Available online 16 January 2017

Keywords:
Real-time scheduling
Cyber-physical systems
Periodic real-time task model
Task-level fixed-priority scheduling
Priority assignment

ABSTRACT

In a typical cyber-physical system (CPS), the cyber/computation subsystem controls the physical subsystem, and therefore the computer society has recently paid considerable attention to CPS research. To keep such a CPS stable, feedback control with periodic computation tasks has been widely used, and its theoretical guarantee of stability has been made with periodic real-time task models that enforce strict periodic control updates. However, some control update misses are usually allowed (e.g., via system over-design) in certain physical subsystem states (PSSes) without causing system instability, and the resources required for strict periodic control updates can thus be reduced or used for other purposes, achieving *efficient* controls for the entire CPS in terms of the operational cost, such as fuel consumption or tracking accuracy. In this paper, we propose a new *periodic, fault-tolerant CPS task model*, which not only expresses efficiency and stability of the underlying physical subsystem, but also generalizes existing periodic real-time task models, by capturing a tolerable number of control update misses in different PSSes. To demonstrate the utility of this model, we develop a new scheduling mechanism that prioritizes jobs (i.e., periodic invocations) of a set of tasks not only by the nature of each task, but also by the number of consecutive prior job deadline misses. Based on its analysis in terms of stability and efficiency, we also propose a priority-assignment policy that lowers the system operation cost without compromising stability. Our in-depth analysis and simulation results show that the scheduling mechanism and its analysis, as well as the priority-assignment policy under the proposed model not only generalize the existing periodic real-time task models, but also significantly lower the system operation cost without losing stability.

1. Introduction

Engineered systems with computation and physical parts have been getting more complex and tightly interacting with each other, necessitating a new concept of Cyber-Physical Systems (CPSes) (National Science Foundation NSF, 2014; Lee, 2006, 2008). As a new systems science of tight integration and coordination between computation and physical processes, CPS researchers have been exploring various functional/non-functional attributes, such as stability, efficiency, reliability, predictability, and autonomy. Since the computation process takes an important part in CPSes, the computer society has paid attention to CPS research, e.g., Chipara et al. (2013); Shen et al. (2015); Yang et al. (2015).

In a diverse collection of CPSes, such as airplanes, robots, and automobiles, the computation part *controls* the physical part, and such control systems have traditionally focused on stability—how to keep the systems stable to prevent catastrophic failure or crash. To achieve stability, feedback control with periodic computation tasks has been widely studied and deployed in real systems. That is, each task periodically executes its control "jobs"[1] with input data taken at the beginning of a period, and as long as every job completes execution within the period, the feedback controller guarantees system stability. Based on periodic control task models (Liu and Layland, 1973), numerous real-time scheduling algorithms have also been developed to make such deadline guarantees with limited resources (Törngren, 1998; Davis and Burns, 2011). These studies presume that *all* control jobs must be completed before their deadlines, but this is usually not strictly required in real

* Corresponding author.
E-mail addresses: jinkyu.lee@skku.edu (J. Lee), kgshin@eecs.umich.edu (K.G. Shin).

[1] In the field of real-time computing, periodic invocations of a computation *task* are called *jobs*.

http://dx.doi.org/10.1016/j.jss.2017.01.004
0164-1212/

(a) Even, almost straight road, e.g., highway: fewer control updates needed

(b) Unpaved, winding road, e.g., off-road: more frequent control updates needed

Fig. 1. Periodic steering wheel controls for an unmanned ground vehicle under different physical subsystem's states.

deployed systems. For example, when an unmanned ground vehicle is running on an unpaved, winding road as shown in Fig. 1(b), strictly periodic control updates of the steering wheel are necessary for the vehicle to follow the road without veering off the road, but some control update misses may not compromise the vehicle's safety/stability when the vehicle is running on an even, almost straight road as in Fig. 1(a). In this case, the resources allocated for the strictly periodic controls of the steering wheel can be reduced or used to finish other control tasks (e.g., controls of acceleration and break pedals) earlier, achieving more efficient controls for the entire system in terms of operational or control cost, such as fuel consumption or deviation from the desired vehicle's trajectory.

The goal of this paper is to achieve efficient controls without compromising stability for CPSes with limited resources by developing a new task model and a scheduling mechanism that satisfy the following requirements *together*:

- G1 A new task model should express stability and efficiency (i.e., control cost) by capturing the number of tolerable control update misses; and
- G2 A new scheduling mechanism, associated with the new task model, should reduce the control cost while guaranteeing system stability.

We first develop a new task model that meets G1. To abstract the effect of a control task on stability and efficiency, the model employs the number of consecutive job deadline misses of a given task, i.e., the frequency of control updates. Then, the model is used to address stability and efficiency, respectively, by limiting the number (i.e., the minimum frequency) of control updates for each task, and mapping that number to the control cost (i.e., specifying the control cost according to the frequency of control updates). Note that since both the limit and the mapping vary with different *physical subsystem's states* (PSSes), e.g., the condition of a road on which the vehicle is running in Fig. 1(a) and (b), the model differentiates them according to PSSes. We call this model the *periodic, fault-tolerant CPS task model* which will be detailed in Section 2: the definition and properties, and the necessity of the new model compared to existing task models.

To devise a new scheduling mechanism that meets G2, we differentiate jobs of a periodic task by the number of consecutive job deadline misses of the task experienced before, which is a key parameter for stability and efficiency under the periodic fault-tolerant CPS task model. We call the number a *cyber subsystem's state* (CSS), and develop a *CSS-level fixed priority scheduling algorithm* (CFP), which prioritizes jobs not only by the task itself that invokes them, but also by each job's CSS. For a given CSS-level fixed priority ordering, we perform a stability analysis—whether jobs of each task exceed the maximum tolerable number of consecutive job deadline misses, and efficiency—the control cost by each task according to the actual number of consecutive deadline misses of its jobs. Using this analysis, we develop an algorithm that finds a CSS-level priority ordering that can lower the control cost without causing system instability. To this end, we identify useful properties of the optimal priority ordering, and incorporate them into a greedy approach to reducing time-complexity. We prove that CFP, its analysis, and the proposed priority assignment are generalizations of the task-level fixed priority scheduling algorithm (TFP) (Liu and Layland, 1973), its analysis (Audsley et al., 1991), and its optimal priority assignment (Audsley, 1991), associated with the traditional periodic task model (Liu and Layland, 1973). Also, our in-depth evaluation results show that CFP with this priority assignment significantly reduces the control cost while guaranteeing system stability, compared to the optimal priority assignment for TFP with the traditional periodic task model and an extension thereof (Buttazzo et al., 1998, 2002).

The novelty of this paper is to address G1 and G2 *together*. While there are some existing task models which address a part of G1 (See Section 2), they did not properly abstract the efficiency in G1 so as to facilitate development of G2. In summary, this paper makes the following two significant contributions:

- Development of a new periodic, fault-tolerant CPS task model that not only addresses stability and efficiency in different PSSes, but also generalizes existing task models; and
- Development of CFP, its analysis and priority assignment, which not only are generalizations of the corresponding scheduling, analysis and priority assignment, but also significantly reduce the control cost without losing stability, thus demonstrating benefits of the periodic fault-tolerant CPS task model.

The remainder of the paper is organized as follows. Section 2 identifies the requirements of a CPS task model, and develops a new CPS task model to meet the requirements. Section 3 makes a formal definition of CFP, while Section 4 analyzes it in terms of stability and efficiency. Based on the analysis, Section 5 derives a priority-assignment policy towards the optimal priority ordering. Section 6 evaluates the efficiency and stability of CFP under the proposed priority-assignment policy. Finally, Section 7 concludes the paper.

2. A new periodic CPS task model

In this section, we develop a new task model that captures CPS stability and efficiency. We first discuss the requirements of a CPS task model and elaborate why existing task models cannot fully meet these requirements. Then, we introduce a new CPS task model and show how it meets the CPS requirements, along with its salient properties.

2.1. Requirements of a new CPS task model

The periodic task model in Liu and Layland (1973) has been widely used to represent the feedback loop of a CPS where computation tasks (part of the cyber subsystem) control the physical subsystem, or the controlled plant. In that model, a real-time control task $\tau_i \in \tau$ is specified with its period T_i and worst-case computation time C_i. That is, at every T_i time units, an invocation/job of τ_i is released with input data (e.g., from sensors) taken at the beginning of the period; each job takes at most C_i time units for control computation, and must be completed before the release of τ_i's next job, i.e., each job of τ_i is completed (discarded) within (after) T_i time units. Let $J_{i,h}$ be the *hth* job of τ_i. The execution window

of $J_{i,h}$ represents an interval between the release and deadline of $J_{i,h}$, and $|\tau|$ denotes the number of tasks in τ.

The periodic task model is used to achieve the physical subsystem's stability and/or other objectives by enforcing every job to complete before its deadline. However, such a stringent requirement is usually overly conservative/pessimistic since real (physical) systems in some PSSes (e.g., Fig. 1(a)) have built-in margins of "safety" and hence do not lose stability even if some jobs miss their deadlines. However, a certain number of job deadline misses degrade control performance/cost, such as fuel/time consumption or deviation from a desired trajectory; the more job deadline misses, the more control cost. This calls for a more general task model (than existing ones) that captures tolerable job deadline misses and the resultant control cost (related to CPS efficiency). To capture the stability and efficiency of CPSes, the model should meet the following requirements.

R1 The new CPS task model should capture tolerable job deadline misses without losing system stability.
R2 The model should capture the control cost associated with job deadline misses (that do not cause system instability).
R3 The model should express a number of job deadline misses with finite (and meaningful) states, such that the states capture the coupling between cyber and physical subsystems.

While R1 and R2 deal with the model's expressiveness in terms of CPS stability and efficiency, respectively, R3 enables the model to provide a manageable interface/abstraction between cyber and physical subsystems that will enable the development of a scheduling mechanism. That is, under the model compliant with R3, one can (re)arrange schedules that yield less control cost without losing stability, by identifying and prioritizing tasks using their abstracted states.

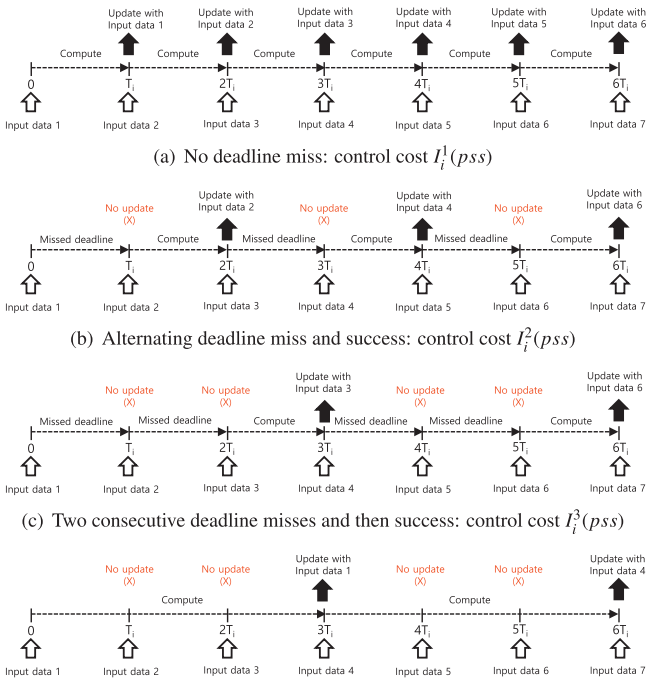
2.2. Why not existing task models?

Before proposing a new CPS task model that satisfies R1–R3, we would like to review if existing models can meet R1–R3. Among the numerous models of periodic control/non-control tasks, we first discuss existing models that allow some job deadline misses. We can classify such fault-tolerance models with the following two notions, as defined in Bernat et al. (2001).

- Of M consecutive jobs of a task, at least N jobs in any order meet their deadlines, such as (i) $(m-k)$ deadline (Hamdaoui and Ramanathan, 1995; Ramanathan, 1999) with $N=m$ and $M=k$; (ii) the skip factor s (Kohen and Shasha, 1995) with $N=s$ and $M=(s+1)$; (iii) window-constrained model (West et al., 2004) with $N=x$ $M=y$; and (iv) job skipping (Yoshimoto and Ushio, 2011).
- The event of L consecutive job deadline misses never occurs, e.g., the criticality factor c and sensitivity of s (Wedde and Lind, 1997).

Although the fault-tolerance task models can meet R1 by limiting tolerable job deadline misses with these two notions, they cannot satisfy R2 because they do not account for the control cost associated with a number of job deadline misses. Also, they do not define/provide the concept of meaningful finite states that capture the coupling between cyber and physical subsystems described in R3. There are also some models that allow deadline misses (Fontanelli et al., 2013a, 2013b), but they also do not fully address R2 and R3. Therefore, we need a more general task model that captures both the stability and efficiency of CPSes.

Besides the fault-tolerance models, there are other task models proposed to accommodate more general task parameters, such as elastic task parameters (Buttazzo et al., 1998, 2002; Chantem et al., 2009), continually varying periods (Kim et al., 2012), two or more phases of the execution time (Mok and Chen, 1997; de Niz et al., 2012), mandatory and optional execution parts (Aydin et al., 2001), and control update delay analysis (Kim and Shin, 1994). Since these task models do not allow any deadline miss, they capture neither efficiency (i.e., R2) nor the coupling between cyber and physical subsystems with finite states (i.e., R3).



(a) No deadline miss: control cost $I_i^1(pss)$

(b) Alternating deadline miss and success: control cost $I_i^2(pss)$

(c) Two consecutive deadline misses and then success: control cost $I_i^3(pss)$

(d) Period extension by three: control cost larger than the case of (c) due to use of an old input data

Fig. 2. Control cost by different update frequencies, where Update i is the output computed using Input data i.

2.3. New periodic CPS task model

To develop a new periodic, fault-tolerant CPS task model satisfying R1–R3, we make the following two observations.

O1 The stability of CPSes depends on the number of consecutive control update misses. In other words, a control system does not lose its stability in the presence of up to X consecutive control update misses, where X depends on the dynamics and state of the underlying control system.
O2 The control cost depends on the frequency of control updates; the higher the update frequency, the less the control cost. For example, the control cost with no deadline miss in Fig. 2(a) (i.e., update period of T_i) is smaller than that with alternating deadline miss and success in Fig. 2(b) (i.e., update period $= 2 \cdot T_i$), which is also smaller than that with two consecutive deadline misses and then success in Fig. 2(c) (i.e., update period $= 3 \cdot T_i$).

Based on O1 and O2, we abstract the number of job deadline misses with the number of *consecutive* job deadline misses. While this abstraction has a direct connection to R1 as shown in O1, it also satisfies R2. For example, the control cost when there is at most one (likewise two) deadline miss(es) after a successful control update is upper-bounded by that with alternating deadline miss and success in Fig. 2(b) (likewise two consecutive deadline misses and then success in Fig. 2(c)). Therefore, while the control cost depends on the update frequency as mentioned in O2, we

can express an upper-bound of the control cost by associating the control cost with the number of consecutive job deadline misses, hence satisfying R2.

Therefore, we propose a new model that captures the stability and efficiency of a CPS via the *number of consecutive job deadline misses* for each real-time control task. To specify the control update requirements for stability in different PSSes, we use $m_i(pss)$ to denote the maximum number of consecutive job deadline misses of task τ_i without losing stability in a given PSS. Also, to identify jobs of τ_i according to the number of consecutive job deadline misses, let $J_{i,h}^{\ell}$ $(1 \le \ell \le m_i(pss)+1)$ denote the *hth* job of τ_i after missing $(\ell-1)$ consecutive job deadlines, which is defined mathematically as:

$$J_{i,h}^{\ell} \triangleq J_{i,h} | J_{i,h-1}, \cdots, J_{i,h-\ell+1} \text{ miss their job deadlines while } J_{i,h-\ell} \text{ does not.} \quad (1)$$

For notational simplicity, we omit h whenever it is irrelevant, e.g., J_i^{ℓ} instead of $J_{i,h}^{\ell}$. We call ℓ in J_i^{ℓ} the *cyber subsystem state* (CSS), and express J_i^{ℓ} as a job of τ_i with CSS ℓ. Then, τ_i has $(m_i(pss)+1)$ different classes of jobs according to CSSes, i.e., $\ell = 1, 2, \cdots, m_i(pss)+1$.

To express the control cost for each job with CSS ℓ, we let $I_i^{\ell}(pss)$ denote the control cost incurred by τ_i when the control is updated by τ_i every $\ell \cdot T_i$ under a given PSS. For example, $I_i^1(pss)$, $I_i^2(pss)$ and $I_i^3(pss)$ represent the control costs under a given PSS, when there are 0, 1 and 2 consecutive deadline misses after each successful control update, respectively, as shown in Fig. 2(a), (b) and (c). Note that the control costs $\{I_i^{\ell}(pss)\}$ are associated with *fresh input data*; for example, Updates 3 and 6 in Fig. 2(c) are made with Input data 3 and 6, respectively, and therefore, $I_i^3(pss)$ in Fig. 2(c) is smaller than the control cost resulting from use of an *old input data*, i.e., the control cost with Updates 1 and 4 in Fig. 2(d), which are made with Input data 1 and 4, not with Input data 3 and 6.

Obviously, $I_i^{\ell}(pss)$ is a non-decreasing function of ℓ; by definition of $m_i(pss)$, $I_i^{\ell}(pss) = \infty, \forall \ell > m_i(pss)+1$. Also, the control cost incurred by τ_i does not exceed $I_i^X(pss)$ for given $1 \le X \le m_i(pss)+1$, as long as the actual number of consecutive job deadline misses of τ_i is less than X.

The proposed CPS task model satisfies R3 in that the number of consecutive job deadline misses not only provides the task model with a "good" abstraction of job deadline misses, but also enables the model to capture the coupling between cyber (ℓ) and physical subsystems (stability and efficiency). That is, as long as every job of τ_i associated with a cyber element ℓ (i.e., J_i^{ℓ}) does not miss its deadline for a given $1 \le \ell \le m_i(pss)+1$, we can meet the efficiency and stability requirements of physical subsystems, by guaranteeing that τ_i does not incur more than the control cost $I_i^{\ell}(pss)$, and τ_i does not cause the underlying system to be unstable.

We call the new task model the *periodic, fault-tolerant CPS task model* in contrast to the periodic *pure* task model (Liu and Layland, 1973). Note that one may argue that the model of (m-k) deadline (Hamdaoui and Ramanathan, 1995; Ramanathan, 1999) can express what our task model does, which is not true; for better understanding, we will give an example regarding the difference between the two models at the end of Section 6. In this paper, we would like to capture and address both the stability and efficiency in a given PSS. For simplicity of presentation, we omit the variable PSS for relevant parameters, i.e., using m_i and I_i^{ℓ} instead of $m_i(pss)$ and $I_i^{\ell}(pss)$. Then, the periodic fault-tolerant CPS task model has the following properties.

- $J_{i,h}^1$ *exists* only if $J_{i,h-1}$ completes its execution within T_i time units, or $h = 1$, i.e., the first job of τ_i. On the other hand, $J_{i,h}^{\ell}$ for $2 \le \ell \le m_i+1$ *exists* only if there exists a predecessor job $J_{i,h-1}^{\ell-1}$ and it misses its deadline.
- If $J_{i,h}^{\ell}$ does not finish its execution within T_i time units, the job's control update will not be made and the system will use a default value, such as the previous/old value, depending on the underlying control algorithm. Thus, we do not execute the job after its deadline, meaning that at most one job per periodic task will be ready to execute at any time.

The periodic fault-tolerant CPS task model is more general than the existing periodic pure task model, as stated in the following lemma.

Lemma 1. *The periodic fault-tolerant CPS task model subsumes the periodic pure task model (Liu and Layland, 1973).*

Proof. If we assign $m_i = 0$ for all $\tau_i \in \tau$, the periodic fault-tolerant CPS task model becomes the usual real-time task model. □

Now, we summarize and emphasize the distinct characteristics of the periodic fault-tolerant CPS task model.

C1 The model can express both the stability and efficiency of a control system.

This has not been achieved by any other existing periodic task models, including the fault-tolerance models.

C2 The model does not change the sampling frequency, and provides a more general form of control updates.

One approach to adjusting the control update frequency is to change the sampling frequency, e.g., (Seto et al., 1996), which can be accommodated by the task models that adjust task periods (Buttazzo et al., 1998, 2002; Chantem et al., 2009; Kim et al., 2012), or by overrun execution (Cervin, 2005). While this approach is suitable for some systems, e.g., automotive engine control (Kim et al., 2012), it does not have general applicability because changing the sampling frequency yields different physical behavior than allowing some job deadline misses. To see this difference, let us consider two cases when (i) the task period is extended by a factor of 3 as shown in Fig. 2(d); and (ii) two consecutive deadline misses are allowed under our new task model. If the number of actual consecutive deadline misses is 2 for (ii) as shown in Fig. 2(c), the update periods of both (i) and (ii) would be the same as, and equal to $3 \cdot T_i$, where T_i is the original task period. However, the accuracy of each control update is different because (i) and (ii) use, respectively, $3 \cdot T_i$ and T_i time units old inputs as shown in Figs. 2(d) and (c). Besides the accuracy difference due to different input data, if there is no or only one deadline miss after a successful control update for (ii) as shown in Fig. 2(a) or (b), then (ii) results in a higher update frequency than (i), because (i) can update its control every $3 \cdot T_i$ time units while (ii) can try a control update at every T_i or $2 \cdot T_i$. This is a distinct feature of our task model, which is provided by neither the task models with frequency change (Buttazzo et al., 1998, 2002; Chantem et al., 2009; Kim et al., 2012) nor those with self-triggered execution (Velasco et al., 2003; Anta and Tabuada, 2009). Therefore, our model expresses more accurate controls and offers a more general form of control updates.

C3 The characteristics of C1 and C2 are abstracted by a single parameter (ℓ), the number of consecutive job deadline misses of a given task.

Thanks to C3, scheduling a control task set associated with our task model can easily address stability and efficiency by differentiating jobs according to the parameter ℓ. We will present how a

scheduling mechanism utilizes such an abstraction of stability and efficiency in Sections 3, 4 and 5.

One may argue for the need of capturing the average number of consecutive job deadline misses. However, as discussed so far, guaranteeing the stable and efficient control of CPSes requires the actual (but *not* average) number of consecutive job deadline misses. To see this more clearly, let us consider two cases: (i) alternating deadline miss and success in Fig. 2(b), and (ii) no deadline miss for an arbitrarily long period of time and then no control update for that same period. Then, although the average number of deadline misses of (i) is the same as that of (ii), (ii) causes system instability due to no update for a long period of time while (i) does not. Therefore, the average number of deadline misses is not a suitable abstraction for the control of CPSes.

In summary, the periodic fault-tolerant CPS task model captures not only the stability and efficiency of control systems under different PSSes, but also more general scenarios, only with a single additional parameter. Thus, the model is well-suited for real-time control tasks in CPSes.

3. CSS-level fixed priority scheduling

Suppose, for example, that in a given PSS, up to two deadline misses are tolerable after a successful control update for τ_k (i.e., $m_k = 2$). Then, there are three types of τ_k's jobs according to their CSSes: J_k^1, J_k^2 and J_k^3. The timely execution of J_k^3 is more important/critical than that of J_k^1 and J_k^2 since missing J_k^3's deadline leads to system instability while missing J_k^1's and J_k^2's does not. Of J_k^1 and J_k^2, missing the deadline of J_k^2 not only incurs more control cost (i.e., $I_k^1 \leq I_k^2$), but also leads to transitioning to a more critical job state (i.e., J_k^3). Therefore, in scheduling a job of τ_k with other ready jobs, the criticality of τ_k's job should be considered differently according to its CSS, thus calling for the need to assign priority according to not only the task itself that invokes jobs, but also each job's CSS. In this section, we describe the CSS-level fixed priority scheduling algorithm and useful properties thereof.

Let p_k^ℓ ($\tau_k \in \tau$, $1 \leq \ell \leq m_k + 1$) denote the global CSS-level fixed priority of J_k^ℓ; a larger index indicates a higher priority for jobs of τ_k with the cyber subsystem state ℓ. We consider a scheduling algorithm which always executes a job J_k^ℓ with the highest CSS-level priority (p_k^ℓ) among all ready jobs. We call this scheduling the *CSS-level fixed priority* (CFP) scheduling which is described formally in Algorithm 1. While the job release and completion processes in the algorithm are similar to those of the task-level fixed priority scheduling (TFP) (Liu and Layland, 1973), the timer expiration process represents the property of the new task model—disallowance of the execution of a job J_k^ℓ that does not finish execution within T_k time units. Note that Algorithm 1 as well as its analysis and priority assignment to be presented in Sections 4 and 5 target a uniprocessor platform, which can serve at most one job at a time.

Note that CFP is a new type of dynamic-priority scheduling algorithm; the task priority changes dynamically according to its jobs' CSS, distinguishing it from the traditional dynamic-priority scheduling algorithms (e.g., EDF) in which the task priority varies with its jobs' deadlines.

CFP is a generalization of TFP, as stated in the following lemma.

Lemma 2. *The CFP scheduling algorithm subsumes the TFP scheduling algorithm (Liu and Layland, 1973).*

Proof. If we assign a task-level uniform priority to all jobs of $\tau_k \in \tau$, i.e., $\{p_k^\ell = p_k\}_{1 \leq \ell \leq m_k+1}$, CFP with the periodic CPS task model is equivalent to TFP with the periodic task model. □

Algorithm 1 CFP scheduling algorithm.

Job release: when J_k^ℓ is released,

```
Set the timer of J_k^ℓ to t + T_k, where t is the current time.
Check whether there is a currently executing job J_i^y.
if J_i^y does not exist or p_i^y < p_k^ℓ holds then
    if J_i^y exists then
        Stop the execution of J_i^y and put the job to the ready queue
    end if
    Start the execution of J_k^ℓ.
else
    Put J_k^ℓ in the ready queue.
end if
```

Job completion: when the currently executing job J_i^y finishes its execution,

```
Unset the timer of J_i^y
if the ready queue is not empty then
    Start the execution of a job with the highest priorityin the ready queue.
end if
```

Timer expiration: when the timer of a job J_i^y expires,

```
if J_i^y is currently executing then
    Stop the execution of J_i^y, and perform the job completion process.
else
    Remove J_i^y from the ready queue.
end if
```

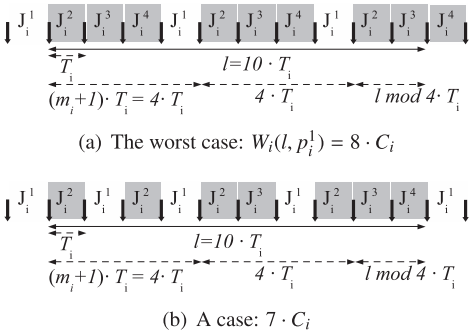
4. Analysis of stability and efficiency for CFP

Before deploying and operating a system in the field, we need to guarantee offline that (i) the system will always remain stable and (ii) controlling the system will not cost more than budgeted (e.g., amount of reserved fuel). In this section, we analyze requirements (i) and (ii) when a task set is scheduled by CFP with given CSS-level fixed priority. While the first analysis generalizes the analysis of TFP, our (improved) analysis links the deadline miss conditions of the previous jobs to those of the job of interest, providing tighter results. This analysis will be used as a basis for better priority assignment to reduce operational cost in (ii) while guaranteeing (i), as detailed in Section 5.

4.1. The first analysis of CFP

The analyses of existing scheduling algorithms associated with the periodic pure task model (Liu and Layland, 1973) usually focus on stability—meeting all job deadlines. To analyze both stability and efficiency, we need to adapt the analysis for stability with respect to the periodic fault-tolerant CPS task model and extend it for efficiency. To this end, we express stability and efficiency using the response time of each job J_k^ℓ, measured from its release to completion. That is, if the response time of J_k^ℓ does not exceed T_k, we guarantee that (i) $J_k^{\ell+1}$ does not exist, and (ii) the control cost incurred by τ_k does not exceed I_k^ℓ, which address stability and efficiency, respectively.

Now, we calculate an upper-bound of a job's response time. Since the response time of J_k^ℓ depends on the execution time of other higher-priority jobs, we first compute the amount of time jobs of $\tau_i \in \tau$ to execute in the execution window of J_k^ℓ. Let $W_i(l, p)$ denote an upper-bound of the amount of execution of τ_i's jobs with priority strictly higher than p in an interval of length l such



(a) The worst case: $W_i(l, p_i^1) = 8 \cdot C_i$

(b) A case: $7 \cdot C_i$

Fig. 3. The amount of execution of τ_i's jobs with priority higher than p_i^1 in an interval of length l such that the interval starts at one of the release times of jobs of τ_i.

that the interval starts at one of the release times of τ_i's jobs. To calculate $W_i(l, p)$, let $n_i(p)$ denote the number of τ_i's CSSes whose priorities are strictly higher than p; by definition, the range of $n_i(p)$ is $[0, m_i + 1]$. Then, as shown in Fig. 3(a) where $m_i = 3$ and $l = 10 \cdot T_i$, for every $(m_i + 1) \cdot T_i$ time units, there exist at most $n_i(p_i^1) = 3$ jobs of τ_i, with priority higher than p_i^1. Therefore, at most $8 \cdot C_i$ amount of execution is done by J_i^2, J_i^3 and J_i^4 (i.e., jobs of τ_i with priority higher than p_i^1) during an interval of length $l = 10 \cdot T_i$ that starts at one of the release times of jobs of τ_i. However, this worst case does not always occur; if J_i^2 (J_i^3) finishes execution within T_i time units, J_i^3 (J_i^4) does not exist as shown in Fig. 3(b), resulting in only $7 \cdot C_i$ amount of execution done by J_i^2, J_i^3 and J_i^4 in an interval of length $l = 10 \cdot T_i$. Considering the remaining part which does not belong to the multiple of $(m_i + 1) \cdot T_i$ as shown in Fig. 3, we calculate $W_i(l, p)$ as follows.

$$W_i(l, p) = \left\lfloor \frac{l}{(m_i+1) \cdot T_i} \right\rfloor \cdot n_i(p) \cdot C_i + \min\left(\left\lceil \frac{l \bmod (m_i+1) \cdot T_i}{T_i} \right\rceil, n_i(p) \right) \cdot C_i. \quad (2)$$

Using $W_i(l, p)$, the following lemma derives a property of the execution time of jobs that have higher priority than p.

Lemma 3. *The time of executing jobs of tasks in τ that have higher priority than p in an interval of length l is upper-bounded by*

$$\sum_{\tau_i \in \tau} W_i(l, p). \quad (3)$$

Proof. We consider two cases: (a) every $\tau_i \in \tau$ with $W_i(l, p) > 0$ satisfies that one of jobs of τ_i is released at the beginning of the interval; (b) otherwise.

By definition, $W_i(l, p)$ is an upper-bound of the execution time of τ_i's jobs with higher priority than p in Case (a). Therefore, the lemma of Case (a) follows, if Eq. (2) is correct, which is proved as follows.

For a job of τ_i to finish its execution, we need 1, 2, ... , or $m_i + 1$ tries, in which the number of jobs whose priority is higher than p is 1 - $(m_i + 1 - n_i(p))$, 2-$(m_i + 1 - n_i(p))$, ... or $n_i(p)$, respectively. For example, in the previous example for Fig. 3, if there is 1, 2, 3 and 4 tries for a job of τ_i to finish its execution, the number of jobs whose priority is higher than p_i^1 is 0, 1, 2 and 3, respectively. Therefore, to have more jobs whose priority is higher than p within the interval of interest, (i) the interval should start at the release of J_i^x with the smallest x such that J_i^x has a higher priority than p, as illustrated in Fig. 3(a). Also, to have more jobs whose priority is higher than p within the interval of interest, (ii) each job finishes its execution with the maximum try, as illustrated in Fig. 3(a) as opposed to Fig. 3(b). Therefore, what remains is to upper-bound the amount of execution of jobs whose priority is higher than p within the interval of length ℓ, when (i) and (ii) are satisfied, which is calculated by Eq. (2) as follows. In the inequality, $\lfloor \frac{l}{(m_i+1) \cdot T_i} \rfloor$ in the first-term computes the number of full cycles (each of which consists of J_i^2, J_i^3, J_i^4, and J_i^1 in Fig. 3(a)) of jobs when (i) and (ii) are satisfied; then, the first term means the amount of execution whose priority is higher than p from the full cycles. For example, there are two full cycles within the interval of interest of length $\ell = 10 \cdot T_i$ in Fig. 3(a). Next, the second term means the amount of execution of jobs (whose priority is higher than p) that do not belong the full cycles within the interval of length ℓ; for example, such jobs are the last two jobs J_i^2 and J_i^3 in the interval of interest of length $\ell = 10 \cdot T_i$ in Fig. 3(a). Therefore, Eq. (2) is an upper-bound of the execution time of τ_i's jobs whose priority is higher than p in an interval of length ℓ in Case (a).

In Case (b), suppose we are interested in an interval $[t, t + l)$, a job of $\tau_j \in \tau$ is released at $t - x$ ($0 < x < T_j$), and the amount of execution of τ_j's jobs with priority higher than p in $[t, t + l)$ is strictly larger than $W_j(l, p)$. This means that the job of τ_j released at $t - x$ is not executed fully or partially in $[t - x, t)$ due to the execution of other higher-priority jobs, and the interval $[t - x, t)$ is completely occupied by the job of τ_j or jobs whose priorities are higher than p; otherwise, the amount of execution of τ_j's jobs with priority higher than p in $[t, t + l)$ cannot be larger than $W_j(l, p)$. In this case, if we consider $[t - x, t - x + l)$ instead of $[t, t + l)$, the execution time of jobs of tasks in τ with higher priority than p in $[t - x, t - x + l)$ is larger than or equal to that in $[t, t + l)$. This interval shift can be repeated until the final interval belongs to Case (a) or there is no task τ_j whose jobs' execution with higher priority than p in the interval of interest is strictly larger than $W_j(l, p)$. Then, the execution time of jobs of tasks in τ that have higher priority than p in the final interval is not smaller than that in the original interval. By Case (a), this proves the lemma of Case (b). □

Using the lemma, we can iteratively calculate an upper-bound of the response time of J_k^ℓ, which is similar to the response time analysis under TFP (Audsley et al., 1991).

Theorem 1. *Suppose that a task set τ is scheduled by CFP. Then, $R_k^{x^*}$ is an upper-bound of the response time of J_k^ℓ such that the following formula satisfies $R_k^{x^*+1} \le R_k^{x^*}$, starting from $R_k^0 = C_k$:*

$$R^{x+1} \leftarrow C_k + \sum_{\tau_i \in \tau - \{\tau_k\}} W_i(R^x, p_k^\ell). \quad (4)$$

Proof. Suppose that $R \ge C_k + \sum_{\tau_i \in \tau - \{\tau_k\}} W_i(R, p_k^\ell)$ holds for given R ($\le T_k$); we will prove that R is an upper-bound of the response time of J_k^ℓ. Here, for given J_k^ℓ, we assume that J_k^y with $y > \ell$ does not exist before the job. This is because, if the response time of J_k^ℓ is not larger than T_k, there does not exist J_k^y with $y > \ell$. So, in this computation, J_k^y with $y > \ell$ cannot affect the execution of other jobs with higher priority than J_k^ℓ.

By Lemma 3, the maximum time for J_k^ℓ not to execute in an interval of length R does not exceed $\sum_{\tau_i \in \tau - \{\tau_k\}} W_i(R, p_k^\ell)$. Therefore, if $C_k + \sum_{\tau_i \in \tau - \{\tau_k\}} W_i(R, p_k^\ell)$ is not greater than R, J_k^ℓ's execution (as much as C_k) is finished within an interval of length R. This implies that R is an upper-bound of the response time of J_k^ℓ, which proves the theorem. □

One may argue that the above theorem can be pessimistic because it does not remove $J_i^{\ell+1}$'s effect on Eq. (2) when the response time of J_i^ℓ does not exceed T_i time units, meaning that $J_i^{\ell+1}$ does not exist. The theorem can immediately accommodate this case by assigning the lowest priority to $J_i^{\ell+1}$. However, this is unnecessary since the priority assignment algorithm to be developed in Section 5 will make such an assignment.

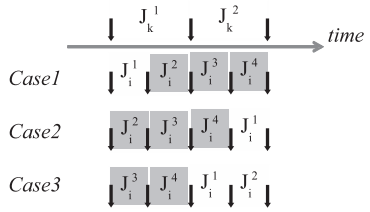


Fig. 4. The worst-case for jobs of τ_i with a higher priority than J_k^1 to execute in the execution window of J_k^1 and that for jobs of τ_i with a higher priority than J_k^2 to execute in the execution window of J_k^2 cannot happen at the same time. Here a grey-colored job of τ_i implies a higher priority than the corresponding job of τ_k, e.g., in Case 1, J_i^2 has a higher priority than J_k^1, and J_i^3 and J_i^4 have a higher priority than J_k^2.

Then, the following lemma states a generalization of the response time analysis in Theorem 1.

Lemma 4. *The response time analysis of CFP associated with the periodic fault-tolerant CPS task model (Theorem 1) is a generalization of the response time analysis of TFP associated with the periodic pure task model (Section 2 in Audsley et al. (1991)).*

Proof. As we proved in Lemma 2, if the task-level uniform priority (p_k) is assigned to J_k^ℓ for all $1 \le \ell \le m_k + 1$, CFP is equivalent to TFP. In such a case, $n_i(p_k)$ is equal to m_i if $p_i > p_k$ (0, otherwise), and therefore, $W_i(l, p_k)$ in Eq. (2) is reduced by $\lceil \frac{l}{T_i} \rceil \cdot C_i$ if $p_i > p_k$ (0, otherwise). Then, Eq. (4) is also reduced by $R^{x+1} \leftarrow C_k + \sum_{\tau_i \in \tau - \{\tau_k\} | p_i > p_k} \lceil \frac{R^x}{T_i} \rceil \cdot C_i$, which is the same as the response time analysis of TFP. □

There are $\sum_{\tau_i \in \tau}(m_i + 1)$ different priorities, and each priority needs the calculation of $|\tau| - 1$ other tasks' W_i for given R_k^x. In addition, at most T_k iterations are required for the calculation of $R_k^{x^*}$. Thus, Theorem 1 incurs $O\big(|\tau| \cdot \sum_{\tau_i \in \tau} m_i \cdot \max_{\tau_i \in \tau} T_i\big)$ time-complexity.

4.2. Improved analysis of CFP

By applying Theorem 1 for all J_i^ℓ ($\tau_i \in \tau$, $1 \le \ell \le m_i + 1$), we know whether a given J_i^ℓ finishes its execution within T_i time units. Although the analysis provides conservative guarantees on stability and the required control cost, it may introduce a non-trivial pessimism in computing the amount of higher-priority job execution. That is, while considering the worst-case situation where the release times of all higher-priority jobs coincide, we can prove that the situation cannot happen for many cases, an example of which is given next.

Example 1. Suppose a task system consists of several task sets including $\tau_k = (T_k = 10, C_k = 5, m_k = 1)$ and $\tau_i = (5, 3, 3)$, and the relative priority ordering for jobs of τ_k and τ_i is $p_i^1 < p_k^1 < p_i^2 < p_k^2 < p_i^3 < p_i^4$. As shown in Fig. 4, $W_i(T_k, p_k^1) = 2 \cdot C_i$ for J_k^1 holds (see Cases 2 and 3). This is because if we focus on J_k^1, J_i^2 and J_i^3, or J_i^3 and J_i^4 can execute by delaying J_k^1's execution. Similarly, $W_i(T_k, p_k^2) = 2 \cdot C_i$ for J_k^2 holds due to the execution of J_i^3 and J_i^4 (see Case 1).

However, to calculate the response time of J_k^2, we can extend the interval of interest to a continuous interval of length $2 \cdot T_k$ including J_k^2 itself and its predecessor job J_k^1, because J_k^2 exists only when J_k^1 misses its deadline. Then, as shown in the same figure, the worst-case for jobs of τ_i with a higher priority than J_k^1 to execute in the execution window of J_k^1 and that for jobs of τ_i with a higher priority than J_k^2 to execute in the execution window of J_k^2 cannot occur at the same time. That is, if jobs of τ_i execute for $2 \cdot C_i$ time units in the execution window of J_k^2, the execution of jobs of τ_i in the execution window of J_k^1 cannot be larger than C_i (see Case 1 in the figure). On the other hand, if jobs of τ_i execute for $2 \cdot C_i$ time units in the execution window of J_k^1, the execution of jobs of τ_i in the execution window of J_k^2 cannot be larger than C_i (see Cases 2 and 3 in the figure). This can lead to timely execution of J_k^2 or the guarantee of non-existence of J_k^2.

As illustrated in the above example, a less pessimistic computation of the response time of J_k^ℓ requires examination of the interval before J_k^ℓ, i.e., $J_k^{\ell-1}$, $J_k^{\ell-2}$, $\cdots$, J_k^1. To investigate the extended interval, we need to identify the worst-case release patterns that upper-bound the execution time of jobs of other tasks in the execution window of jobs of τ_k. However, this is challenging because each predecessor job (e.g., J_k^ℓ, $J_k^{\ell-1}$, $\cdots$, J_k^1) has its own priority, meaning that a set of higher-priority jobs of J_k^ℓ is different from that of $J_k^{\ell'}$ if $\ell \neq \ell'$. To abstract the complex worst-case response time calculation, we derive the following property.

Observation 1. Suppose that $J_k^{\ell-1}$ misses its deadline, and we want to calculate the response time of J_k^ℓ. We consider two cases: (a) J_k^ℓ is executed with its original priority (i.e., p_k^ℓ); and (b) J_k^ℓ is executed with its predecessor priority (i.e., $p_k^{\ell-1}$). Then, the response time of J_k^ℓ in Case (b) is not smaller than that in Case (a).

Since we assign a lower priority to Case (b), the observation is straightforward.

Then, we have new upper-bounds of the response time of J_k^ℓ as follows. While we investigate whether J_k^ℓ finishes its execution (taking C_k time units) within T_k times units in Theorem 1, we investigate 2 consecutive jobs of $J_k^{\ell-1}$ finish their execution (taking $2 \cdot C_k$ time units) within $2 \cdot T_k$ time units, which is generalized in the following theorem.

Theorem 2. *Suppose that a task set τ is scheduled by CFP. Then, an upper bound of the response time of J_k^ℓ is $R_k^{x^*(\alpha)} - T_k \cdot \alpha$ such that the following formula satisfies $R_k^{x^*(\alpha)+1} \le R_k^{x^*(\alpha)}$, starting from $R_k^0 = C_k + T_k \cdot \alpha$, for a given integer α $(0 \le \alpha \le \ell - 1)$:*

$$R^{x+1} \leftarrow (\alpha + 1) \cdot C_k + \sum_{\tau_i \in \tau - \{\tau_k\}} W_i(R^x, p_k^{\ell-\alpha}). \tag{5}$$

Then, we use an upper-bound of the response time of J_k^ℓ as $\min_{0 \le \alpha \le \ell-1} R_k^{x^*(\alpha)} - T_k \cdot \alpha$.

Proof. To calculate the response time of J_k^ℓ, we assume that $J_k^{\ell-1}$ misses its deadline; otherwise, J_k^ℓ does not exist. Using this assumption and Observation 1, we can derive another upper-bound of the response time J_k^ℓ, by calculating the response time of the last job of $J_k^{\ell-\alpha}$ when $\alpha + 1$ consecutive jobs of $J_k^{\ell-\alpha}$ are released and the first α jobs miss their deadlines. □

Note that Theorem 2 is a generalization of Theorem 1 in that the former with $\alpha = 0$ is the same as the latter. Then, by Lemma 4, Theorem 2 is also a generalization of the response time analysis of TFP associated with the periodic pure task model (Section 2 in Audsley et al. (1991)).

Compared to Theorem 1, Theorem 2 repeats the response time calculation process in Theorem 1 at most $\max_{\tau_i \in \tau} m_i$ times. Therefore, the time-complexity of Theorem 2 is $O\big(|\tau| \cdot \sum_{\tau_i \in \tau} m_i \cdot \max_{\tau_i \in \tau} m_i \cdot \max_{\tau_i \in \tau} T_i\big)$.

5. Priority assignment for CFP

In Section 4, we provided a theoretic basis for assuring stability and measuring the required control cost under CFP with a given priority assignment. We now present an algorithm that finds

a "better" priority assignment of CFP, yielding smaller control costs without compromising stability.

Since there are $\sum_{\tau_i \in \tau}(m_i+1)$ different priorities to be assigned for CFP, the number of possible priority assignments is in $O\big((\sum_{\tau_i \in \tau}(m_i+1))!\big)$, which is an exponential function of the number of tasks in τ. Therefore, our goal is to develop an algorithm that finds a "good" priority assignment with reasonable time-complexity. To do this, we design an algorithm that determines priorities from the lowest to the highest. As shown in Algorithm 2, we assign the lowest priority first in Line 1, assuming all undecided priorities are the highest in Line 2. Then, the while-loop in Lines 3–22 is executed until there is no undecided priority. For the given remaining undecided lowest priority, the for-loop in Lines 5–15 investigates a job per task such that the response time of the job does not exceed its period. If so, the current priority is assigned to the job. This preserves the optimal priority assignment in terms of stability and efficiency, as stated in the following observation. Note that $\mathcal{J}$ in Line 16 becomes empty only when $\hat{\ell}$ in Line 7 is equal to m_i+1 and the upper-bound of the response time of $J_i^{m_i+1}$ is larger than T_i (since it does not execute Line 13); this implies the task set's instability.

Observation 2. In Algorithm 2, if the priority of J_i^{ℓ} is assigned by Line 12, J_i^{ℓ} for all $\hat{\ell} \le \ell \le m_i+1$ does not affect the response time of all other jobs whose priorities are not yet determined (i.e., all other jobs which will eventually be assigned a higher priority than J_i^{ℓ} for all $\hat{\ell} \le \ell \le m_i+1$).

Algorithm 2 Priority assignment for CFP.

1: $p_{curr} \leftarrow 1$, i.e., we determine the lowest priority first.
2: $p_i^{\ell} \leftarrow p_{max}, \forall J_i^{\ell}$ where $\tau_i \in \tau$ and$1 \le \ell \le m_i+1$.
3: **while** there exists J_i^{ℓ} such that $p_i^{\ell} = p_{max}$ **do**
4: $\quad\mathcal{J} \leftarrow \emptyset$.
5: $\quad$**for** $\forall \tau_i \in \tau$ such that $\exists p_i^{\ell} = p_{max}$ **do**
6: $\quad\quad\hat{\ell} \leftarrow$ the smallest ℓ such that$p_i^{\ell} = p_{max}$.
7: $\quad\quad$**if** $\hat{\ell} < m_i+1$ **then**
8: $\quad\quad\quad\mathcal{J} \leftarrow \mathcal{J} \cup \{J_i^{\hat{\ell}}\}$
9: $\quad\quad$**end if**
10: $\quad\quad$Calculate an upper-bound of the response time of$J_i^{\hat{\ell}}$ in case it has the priority of p_{curr}using Theorem 2.
11: $\quad\quad$**if** the upper-bound is smaller than or equal to T_i **then**
12: $\quad\quad\quad p_i^{\ell} \leftarrow p_{curr}, \forall \hat{\ell} \le \ell \le m_i+1$.
13: $\quad\quad\quad$Exit for-loop and go to Line 21.
14: $\quad\quad$**end if**
15: $\quad$**end for**
16: $\quad$**if** $\mathcal{J} = \emptyset$ **then**
17: $\quad\quad$Return UNSTABLE
18: $\quad$**else**
19: $\quad\quad$Find $J_i^{\ell} \in \mathcal{J}$ which has the smallest$I_i^{\ell+1} - I_i^{\ell}$, and then $p_i^{\ell} \leftarrow p_{curr}$.
20: $\quad$**end if**
21: $\quad p_{curr} \leftarrow p_{curr}+1$.
22: **end while**
23: Return STABLE with $\{p_i^{\ell}\}$.

The above observation holds since if the response time of $J_i^{\hat{\ell}}$ is smaller than or equal to T_i, then J_i^{ℓ} does not exist for all $\hat{\ell}+1 \le \ell \le m_i+1$. Thus, as long as the for-loop finds such $J_i^{\hat{\ell}}$, the priority assignment guarantees the control cost incurred by τ_i not to exceed $I_i^{\hat{\ell}}$ and every job of τ_i does not affect the execution of all jobs whose priorities are not assigned but will be higher than $J_i^{\hat{\ell}}$.

However, if the for-loop cannot find any job that satisfies Line 11, we need to assign the lowest undecided priority (p_{curr}) to a job which cannot be completed before its deadline. Here, we do not select a job with CSS (m_i+1) (i.e., $J_i^{m_i+1}$) for all $\tau_i \in \tau$ since it results in system instability. Then, unlike the priority assignment by Line 12, the selected job with CSS ℓ can affect the execution of other jobs whose priorities are not assigned, because it's successive job with CSS $(\ell+1)$ exists and may eventually have higher priority. Among other jobs whose priorities are undecided, we choose a job J_i^{ℓ} with the smallest $I_i^{\ell+1} - I_i^{\ell}$, meaning the selection of a job that results in the smallest rise of control cost with the current assignment. The heuristic decision is described in Lines 16–20 in Algorithm 2.

Except for the heuristic decision when there is no timely-executable job to be assigned, our priority-assignment algorithm is optimal, as stated in the following lemma.

Lemma 5. *Suppose there exists a priority ordering that guarantees stability by Theorem 2. Then, Algorithm 2 finds a priority ordering which results in the smallest control cost without compromising stability with respect to Theorem 2, if no priority is assigned by Line 19.*

Proof. The lemma holds because Algorithm 2 is a generalization of the optimal priority assignment for TFP (Audsley et al., 1991). That is, if Line 19 is not performed, the priority ordering for CFP is reduced to that for TFP because all jobs invoked by a task have the same priority by Line 12. This also means that if the algorithm returns STABLE in Line 23, the control cost is a simple sum of I_i^1 for all $\tau_i \in \tau$, resulting in the minimum control cost without losing stability. □

While Lemma 5 proves the optimality of Algorithm 2 for trivial cases, Section 6 will demonstrate that Algorithm 2 significantly reduces the control cost without compromising stability, even for general cases where some priorities are assigned by Line 19.

Then, whenever we determine a job priority, the for-loop investigates at most one job per task, executing Theorem 2$O(|\tau|)$ times. Since there are $\sum_{\tau_i \in \tau}(m_i+1)$ priorities to be determined, Algorithm 2 needs to execute Theorem 2$O\big(|\tau| \cdot \sum_{\tau_i \in \tau} m_i\big)$ times. Therefore, the total time-complexity of Algorithm 2 is $O\Big(|\tau|^2 \cdot \big(\sum_{\tau_i \in \tau} m_i\big)^2 \cdot \max_{\tau_i \in \tau} m_i \cdot \max_{\tau_i \in \tau} T_i\Big)$.

6. Evaluation

This section demonstrates the capability of the periodic fault-tolerant CPS task model to make a significant improvement of stability and efficiency. We first describe how to generate task sets, and then compare CFP with the priority assignment in Algorithm 2 associated with the periodic CPS task model, and the corresponding scheduling (i.e., TFP) with the optimal priority assignment associated with the periodic pure task model and its extended model.

6.1. Task set generation

As a basis for task set generation, we adapt the technique in Baker (2005), which has been widely used (Bertogna et al., 2009; Lee et al., 2015). To broaden the spectrum of task sets to be tested, we use 10 different distributions for C_i/T_i: bimodal distributions with p =0.1, 0.3, 0.5, 0.7 or 0.9, where the value of C_i/T_i is uniformly chosen in [0, 0.5) with probability p and in [0.5, 1) with probability $1-p$, and exponential distributions with $1/\lambda$ =0.1, 0.3, 0.5, 0.7 or 0.9, whose probability density function is $\lambda \cdot \exp(-\lambda \cdot x)$. For each task, T_i is uniformly chosen in $[1, T_{max} = 1000)$, and C_i is chosen based on a bimodal or exponential parameter.

For each distribution of C_i/T_i, we repeat the following procedure and generate 1000 task sets, thus yielding a total of 10,000 task sets.

1. Initially, generate a set of two tasks.
2. Include the current task set for evaluation.
3. If the number of tasks in the current task set is not larger than $n_{max} = 10$, add a new task to the current task set and return to Step 2. Otherwise, discard the current task set and return to Step 1.

6.2. Generation of control cost functions

While the parameters generated in Section 6.1 are also required for the pure periodic task model, we need to determine additional parameters for the periodic fault-tolerant CPS task model: $\{m_i\}$ and $\{I_i^\ell\}$. To study the effect of m_i, we consider five different settings for m_i; m_i for every task τ_i in each task set is set to 0,1,2,3 and 4. To investigate how the control cost function affects our scheduling, we consider three different control cost functions for $\{I_i^\ell\}$: exponential, linear and random functions, which are denoted by Exp, Lin and Ran, respectively.

First, an exponential function is a typical form of cost function of real control systems. For example, a bubble control system for modeling the diving control in submarines has been studied in Seto et al. (1996), and J is its performance index (see the equation in page 15 of Seto et al. (1996)). Then, the control cost function is represented by the difference between the actual value $J(f)$ and the optimal control J^*, which is shown to be an exponential function as in Fig. 2 in Seto et al. (1996). We can also find other exponential control cost functions in real control systems (see Fig. 7 in Shin et al. (1985), Fig. 4 in Palopoli et al. (2000) and Fig. 4 in Yoshimoto and Ushio (2011)). To generate an exponential function for $\{I_i^\ell\}$, we uniformly distribute I_i^1 for all $\tau_i \in \tau$ in [1, 1000). Then, we set $I_i^{\ell+1}$ to $2 \cdot I_i^\ell$.

Second, we consider a linear function as a control cost function, by setting I_i^1 for all $\tau_i \in \tau$ to a randomly selected value in [1, 1000), and then I_i^ℓ to $\ell \cdot I_i^1$. Finally, a random function is also considered as a control cost function. To apply a random function to $\{I_i^\ell\}$, we uniformly distribute I_i^1 for all $\tau_i \in \tau$ in [1, 1000). Then, I_i^ℓ for all $\tau_i \in \tau$ and $2 \le \ell \le m_i + 1$ is generated by adding a uniformly random number in [1, 1000) to $I_i^{\ell-1}$.

6.3. Comparison with respect to stability and efficiency

With the generated task sets and cost functions, we compare the following three different scheduling algorithms associated with different task models:

- Ours(m): CFP with the priority assignment in Algorithm 2 associated with the periodic fault-tolerant CPS task model when all tasks in each task set have $m_i = m$;
- Pure: TFP (Liu and Layland, 1973) with the optimal priority assignment (Audsley, 1991), associated with the periodic pure task model (Liu and Layland, 1973), i.e., no deadline miss is allowed; and
- Elas(m): TFP (Liu and Layland, 1973) with the optimal priority assignment (Audsley, 1991), associated with the elastic task model (Buttazzo et al., 1998, 2002; Chantem et al., 2009; Kim et al., 2012), i.e., the periodic pure task model (Liu and Layland, 1973) with extension of individual periods (T_i) by ($m+1$).[2]

[2] Since we use each period of τ_i as $(m+1) \cdot T_i$, Elas(m) does not yield any deadline miss as long as Ours(m) is stable. For comparison between Ours(m) and Elas(m), we represent the control cost of Elas(m) using the fresh input data, e.g., Fig. 2(c).

Table 1
The number of task sets proven stable by Pure and Ours(·).

Task model with scheduling	the number of task sets proven stable
Pure	1906
Ours(0)	1906
Ours(1)	2892
Ours(2)	3201
Ours(3)	3336
Ours(4)	3397

Table 2
Control cost ratio between Ours(m) and Elas(m) with different cost functions.

m	Ours(m) / Elas(m)		
	Exp	Lin	Ran
0	1.0	1.0	1.0
1	0.62	0.62	0.66
2	0.46	0.52	0.56
3	0.38	0.47	0.51
4	0.34	0.44	0.48

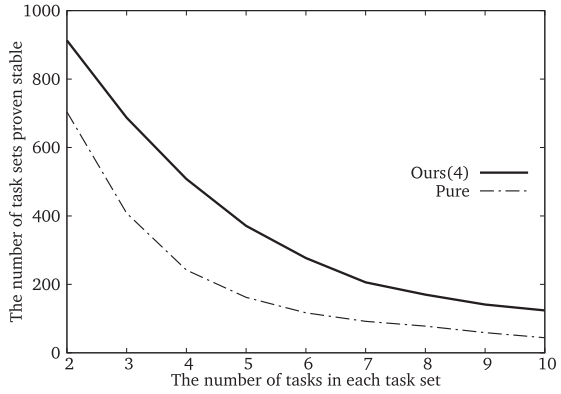
Table 1 shows the number of task sets proven stable by Ours(·) and Pure. Also, Table 2 presents the control cost ratio of Ours(·) to Elas(·), with three different cost functions Exp, Lin, and Ran. From the results in these tables, we identify three different advantages of Ours(·): generalization, stability and efficiency.

First, as we proved earlier, the periodic fault-tolerant CPS task model, and the CFP scheduling algorithm (Algorithm 1) with its analysis (Theorem 2) and priority-assignment algorithm (Algorithm 2) are, respectively, generalizations of the periodic pure task model, and TFP with its analysis and optimal priority-assignment algorithm. Therefore, Ours(0) subsumes Pure as well as Elas(0). The results demonstrate such a relationship; the number of task sets proven stable by Ours(0) and Pure is same in Table 1, and the control cost by Ours(0) is equal to that by Elas(0) in Table 2.

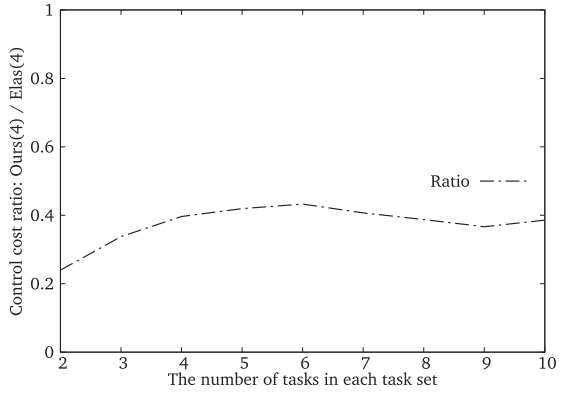
Second, as shown in Table 1, if the current PSS allows more deadline misses, we can accommodate a larger number of tasks. For example, Ours(4) proves additional 78.2% task sets stable, which are not deemed stable by Pure. This improvement is increasing as the number of tasks in each set gets larger, as shown in Fig. 5(a). When the number of tasks is 2, Ours(4) finds 30.0% additional stable task sets, i.e., 913 versus 704; the percentage of additional task sets proven stable by Ours(4) becomes 181.8% when the number of tasks is 10, i.e., 124 versus 44. This is because it is more difficult to guarantee the stability of sets with a larger number of tasks, and thus Ours(4) has much more room to find additional stable sets with more tasks.

Third, as to efficiency, Ours(m) significantly reduces the control cost, compared to Elas(m) as shown in Table 2.[3] The difference between the control cost incurred by Ours(m) and that by Elas(m) increases as the number of allowed consecutive deadline misses (i.e., m) gets larger; when $m = 4$, Ours(4) incurs less than half the control cost of Elas(4) with all cost functions. We also examine how the number of tasks in each task set affects the ratio between the control costs by Ours(4) and Elas(4). As shown in Fig. 5(b),

[3] As we discussed, the control cost of the situation where the control is updated every X periods with fresh input as shown in Fig. 2(c) is no larger than that with old input in Fig. 2(d). Here, we assume both control costs are the same, which gives an advantage to the counterpart Elas(m).



(a) Stability: the number of task sets proven stable by Ours(4) and Pure



(b) Efficiency: control cost ratio between Ours(4) and Elas(4) with Exp

Fig. 5. Comparison of Ours(4) with Pure and Elas(4) in terms of stability and efficiency.

the ratio does not highly depend on the number when the cost function is Exp; the trend is similar with other cost functions.

Since other task models discussed in Section 2.2 have different characteristics for allowing some job deadline misses, we do not directly compare them with our approach. For example, the $(m-k)$ deadline model in Hamdaoui and Ramanathan (1995); Ramanathan (1999) expresses that in any k consecutive execution windows, at least m jobs should meet their deadlines. If we compare the $(m=1, k=3)$ deadline model, with our model with allowing two consecutive deadline misses, the former does not address R2 (i.e., capturing the control cost associated with job deadline misses) and R3 (i.e., expression of a number of job deadline misses with finite states, such that the states capture the coupling between cyber and physical subsystems) in Section 2.1. As shown in the example, our model is more suitable for control systems which should meet Requirements R1–R3 in Section 2.1.

In summary, the periodic fault-tolerant CPS task model and CFP with its analysis and priority assignment not only generalize the existing task model and corresponding scheduling, but also significantly improve stability and efficiency of the system.

7. Conclusion and discussion

In this paper, we have introduced a new periodic fault-tolerant CPS task model, which not only generalizes existing task models, but also expresses the physical system's stability and efficiency by capturing tolerable control update misses. To demonstrate the utility of this model, we have designed and analyzed a new type of dynamic-task-priority scheduling called CFP, and also developed an algorithm for finding a better priority assignment. Our analysis and simulation results have shown that CFP associated with the proposed model is more effective in achieving stable and efficient control of CPSes, than the corresponding scheduling associated with existing models, in that it not only guarantees (offline) more task sets without losing system stability, but also significantly reduces the control cost.

Since our focus was confined to introducing the new periodic fault-tolerant CPS task model and developing a scheduling mechanism with the model under a given PSS, there remains more work to be done for practical use of the model. From the physical subsystem's perspective, we need to develop methods for (i) recognizing the current PSS and transitioning between different PSSes, and (ii) easily and accurately measuring the control cost associated with a given number of consecutive job deadline misses. On the other hand, cyber subsystems need a scheduling mechanism that maintains stability and efficiency during a transition between two different PSSes. Another direction of future work is to study the effect of control update with old data, entailing tradeoff between power saving and control cost.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2014R1A1A1035827), and the Ministry of Education (NRF-2016R1D1A1B03930580). This research was also supported by the MISP (Ministry of Science, ICT & Future Planning), Korea, under the National Program for Excellence in SW) (R2215-16-1005) supervised by the IITP (Institute for Information & communications Technology Promotion). The work reported in this paper was also supported in part by the US Office of Naval Research under Grant No. N00014-15-1-2163, and Global Research Laboratory Program (2013K1A1A2A02078326) through NRF, DGIST Research and Development Program (CPS Global Center) funded by the Ministry of Science, ICT & Future Planning.

References

Anta, A., Tabuada, P., 2009. On the benefits of relaxing the periodicity assumption for networked control systems over CAN. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 3–12.
Audsley, N., 1991. Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times. Technical Report, YCS164. Department of Computer Science, University of York.
Audsley, N., Burns, A., Richardson, M., Wellings, A., 1991. Hard real-time scheduling: the deadline-monotonic approach. In: Proceedings of the IEEE Workshop on Real-Time Operating Systems and Software, pp. 133–137.
Aydin, H., Melhem, R., Mosse, D., Alvarez, P., 2001. Optimal reward-based scheduling for periodic real-time tasks. IEEE Trans. Comput. 50 (2), 111—130.
Baker, T.P., 2005. Comparison of Empirical Success Rates of Global vs. Paritioned Fixed-Priority and EDF Scheduling for Hand Real Time. Technical Report, TR-050601. Dept. of Computer Science, Florida State University, Tallahasee.
Bernat, G., Burns, A., Llamosi, A., 2001. Weakly hard real-time systems. IEEE Trans. Comput. 50 (4), 308–321.
Buttazzo, G.C., Lipari, G., Abeni, L., 1998. Elastic task model for adaptive rate control. In: Proceedings of the 19th IEEE Real-Time Systems Symposium, pp. 286–295.
Buttazzo, G.C., Lipari, G., Caccamo, M., Abeni, L., 2002. Elastic scheduling for flexible workload management. IEEE Trans. Comput. 51 (3), 289–302.
Cervin, A., 2005. Analysis of overrun strategies in periodic control tasks. In: Proceedings of the 16th IFAC World Congress.
Chantem, T., Hu, X.S., Lemmon, M.D., 2009. Generalized elastic scheduling for real–time tasks. IEEE Trans. Comput. 58 (4), 480–495.
Chipara, O., Lu, C., Roman, G.-C., 2013. Real-time query scheduling for wireless sensor networks. IEEE Trans. Comput. 62 (9), 1850–1865.
Davis, R.I., Burns, A., 2011. A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surv. 43, 35:1–35:44.
Fontanelli, D., Palopoli, L., Abeni, L., 2013. The continuous stream model of computation for real-time control. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 150–159.
Fontanelli, D., Palopoli, L., Greco, L., 2013. Optimal cpu allocation to a set of control tasks with soft real-time execution constraints, pp. 233–242.
Hamdaoui, M., Ramanathan, P., 1995. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. IEEE Trans. Comput. 44 (12), 1443–1451.

Kim, H., Shin, K.G., 1994. On the maximum feedback delay in a linear/nonlinear control system with input disturbances caused by controller-computer failures. IEEE Trans. Control Syst. Technol. 2 (2), 110–122.
Kim, J., Lakshmanan, K., Rajkumar, R., 2012. Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems. In: Proceedings of the third IEEE/ACM International Conference on Cyber-Physical Systems, pp. 55–64.
Kohen, G., Shasha, D., 1995. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 110–117.
Lee, E.A., 2006. Cyber-physical systems - are computing foundations adequate? NSF Workshop on Cyber-Physical Systems.
Lee, E.A., 2008. Cyber physical systems: design challenges. In: Proceedings of the 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC), pp. 363–369.
Lee, J., Shin, K.G., Shin, I., Easwaran, A., 2015. Composition of schedulability analyses for real-time multiprocessor systems. IEEE Trans. Comput. 64 (4), 941–954.
Liu, C., Layland, J., 1973. Scheduling algorithms for multi-programming in a hard-real-time environment. J. ACM 20 (1), 46–61.
Mok, A., Chen, D., 1997. A multiframe model for real-time tasks. IEEE Trans. Software Eng. 23 (10), 635–645.
National Science Foundation (NSF), Cyber-physical systems (CPS). 2014, http://www.nsf.gov/pubs/2008/nsf08611/nsf08611.htm [Accessed 10 June.
de Niz, D., Wrage, L., Storer, N., Rowe, A., Rajkumar, R., 2012. On resource overbooking in an unmanned aerial vehicle. In: Proceedings of the third IEEE/ACM International Conference on Cyber-Physical Systems, pp. 97–106.
Palopoli, L., Abeni, L., Buttazzo, G., 2000. Real-time control system analysis: an integrated approach. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 131–140.
Ramanathan, P., 1999. Overload management in real-time control application using (m, k)-firm guarantees. IEEE Trans. Parallel Distrib. Syst. 10 (6), 549–559.
Seto, D., Lehoczky, J.P., Sha, L., Shin, K.G., 1996. On task schedulability in real-time control systems. In: Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 13–21.
Shen, H., Liu, J., Chen, K., Liu, J., Moyer, S., 2015. SCPS: a social-aware distributed cyber-physical human-centric search engine. IEEE Trans. Comput. 64 (2), 518–532.
Shin, K.G., Krishna, C.M., Lee, Y.-H., 1985. A unified method for evaluating real–time computer controllers and its application. IEEE Trans. Automat. Contr. 30 (4), 357–366.
Bertogna, M., Cirinei, M., Lipari, G., 2009. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. IEEE Trans. Parallel Distrib. Syst. 20, 553–566.
Törngren, M., 1998. Fundamentals of implementing real-time control applications in distributed computer systems. Real-Time Syst. 14 (3), 219–250.
Velasco, M., Marti, P., Fuertes, J.M., 2003. The self triggered task model for real–time control systems. In: Proceedings of the Work-in-Progress Session of IEEE Real-Time Systems Symposium (RTSS WiP).
Wedde, H.F., Lind, J.A., 1997. Building large, complex, distributed safety-critical operating systems. Real-Time Syst. 13 (3), 277–302.
West, R., Schwan, K., Poellabauer, C., 2004. Dynamic window-constrained scheduling of real-time streams in media servers. IEEE Trans. Comput. 52 (6), 744–759.
Yang, X., Lin, J., Yu, W., Moulema, P., Fu, X., Zhao, W., 2015. A novel en-route filtering scheme against false data injection attacks in cyber-physical networked systems. IEEE Trans. Comput. 64 (1), 4–18.
Yoshimoto, T., Ushio, T., 2011. Optimal arbitration of control tasks by job skipping in cyber-physical systems. In: Proceedings of the second IEEE/ACM International Conference on Cyber-Physical Systems, pp. 51–64.

Jinkyu Lee is an assistant professor in Department of Computer Science and Engineering, Sungkyunkwan University (SKKU), Republic of Korea, where he joined in 2014. He received the BS, MS, and PhD degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Republic of Korea, in 2004, 2006, and 2011, respectively. He has been a research fellow/visiting scholar in the Department of Electrical Engineering and Computer Science, University of Michigan until 2014. His research interests include system design and analysis with timing guarantees, QoS support, and resource management in real-time embedded systems and cyber-physical systems. He won the best student paper award from the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) in 2011, and the Best Paper Award from the 33rd IEEE Real-Time Systems Symposium (RTSS) in 2012.

Kang G. Shin is the Kevin & Nancy O'Connor Professor of Computer Science in the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor. His current research focuses on QoS-sensitive computing and networking as well as on embedded real-time and cyber-physical systems. He has supervised the completion of 78 PhDs, and authored/coauthored more than 830 technical articles, a textbook and more than 30 patents or invention disclosures, and received numerous best paper awards, including the Best Paper Awards from the 2011 ACM International Conference on Mobile Computing and Networking (MobiCom'11), the 2011 IEEE International Conference on Autonomic Computing, the 2010 and 2000 USENIX Annual Technical Conferences, as well as the 2003 IEEE Communications Society William R. Bennett Prize Paper Award and the 1987 Outstanding IEEE Transactions of Automatic Control Paper Award. He has also received several institutional awards, including the Research Excellence Award in 1989, Outstanding Achievement Award in 1999, Distinguished Faculty Achievement Award in 2001, and Stephen Attwood Award in 2004 from The University of Michigan (the highest honor bestowed to Michigan Engineering faculty); a Distinguished Alumni Award of the College of Engineering, Seoul National University in 2002; 2003 IEEE RTC Technical Achievement Award; and 2006 Ho-Am Prize in Engineering (the highest honor bestowed to Korean-origin engineers). He was a co-founder of a couple of startups and also licensed some of his technologies to industry.